

An efficient quantum circuit analyser on qubits and qudits

T. Loke and J. B. Wang

School of Physics, The University of Western Australia, 6009 Perth, Australia

Abstract

This paper presents a highly efficient decomposition scheme and its associated *Mathematica* notebook for the analysis of complicated quantum circuits comprised of single/multiple qubit and qudit quantum gates. In particular, this scheme reduces the evaluation of multiple unitary gate operations with many conditionals to just two matrix additions, regardless of the number of conditionals or gate dimensions. This improves significantly the capability of a quantum circuit analyser implemented in a classical computer. This is also the first efficient quantum circuit analyser to include qudit quantum logic gates.

1. Program Summary

Title of program: CUGates.m

Programming language used: *Mathematica*

Computers and operating systems: any computer installed with *Mathematica* 6.0 or higher

Distribution format: *Mathematica* notebook

Nature of problem: The *CUGates* notebook simulates arbitrarily complex quantum circuits comprised of single/multiple qubit and qudit quantum gates.

Method of solution: It utilizes an irreducible form of matrix decomposition for a general controlled gate with multiple conditionals and is highly efficient in simulating complex quantum circuits.

Running time: Details of CPU time usage for various example runs are given in Section 4.

Program obtainable from: CPC Program Library, Queens University of Belfast, N. Ireland

2. Introduction

At the heart of a quantum computer lies a set of qubits and/or qudits whose states are manipulated by a series of quantum logic gates, namely a quantum circuit, to provide the ultimate computational results. It is therefore of particular interest to be able to efficiently evaluate the performance of a quantum circuit (such as its reliability, effectiveness, robustness, sensitivity to decoherence and errors) in the design stage using a classical computer.

There are currently several quantum computer simulators reported in the literature [1, 2, 3, 4, 5], which simulate quantum circuits consisting of 1, 2 or 3 qubit gates such as the Hadamard, CNOT and Toffoli gate. The CNOT and Toffoli gate are examples of controlled unitary gates (CUGs), which implement operations that are conditional on the state of the specified control qubits. Other more general CUGs (acting across qubits or qudits) can always be decomposed in terms of a universal set of 1- and 2-qubit quantum gates [6], but this would require significant computational overhead in the analysis. To the best of our knowledge, there are no efficient quantum simulators on quantum circuits with multiple qudit controlled quantum gates.

In this paper, we present a highly efficient scheme for the evaluation of arbitrary CUGs. This scheme reduces the evaluation of multiple unitary gate operations with many conditionals to just two matrix additions, regardless of the number of conditionals or gate dimensions. The implementation of this scheme, and many other functions used to analyse quantum circuits, is provided in a *Mathematica* 7.0 package entitled *CUGates.m*. The computation time required to evaluate the CNOT and Toffoli gates using this package is compared with the *QDENSITY* package [1] and is found to be several orders of magnitude more efficient. Examples of quantum circuits involving controlled unitary gates and their analysis using the notebook are presented. A compilation of the *Mathematica* code presented in the paper is provided in the *Mathematica* notebook *CUGates.nb*.

3. Decomposition of CUGs

3.1. CUGs across qubits

3.1.1. Definitions and notation

Denote a set of qubits as Q , and the wavefunction (if definable) for the i th qubit as $|\psi_i\rangle$. Q is in a basis state iff $|\psi_i\rangle = |0\rangle \vee |\psi_i\rangle = |1\rangle \forall i \in Q$. Define C^{c_i} as being conditional on the state $|1\rangle$ of qubit c_i , and $\bar{C}^{\bar{c}_j}$ as being conditional on the state $|0\rangle$ of qubit \bar{c}_j .

A CUG with conditionals $C^{c_1, \dots, c_i} \bar{C}^{\bar{c}_1, \dots, \bar{c}_j}$ implementing unitary operations $U_1^{u_1} \dots U_k^{u_k}$, where u_1, \dots, u_k denotes the starting qubit of the corresponding U block, is represented by $C^{c_1, \dots, c_i} \bar{C}^{\bar{c}_1, \dots, \bar{c}_j} U_1^{u_1} \dots U_k^{u_k}$. Effectively, the action of this CUG is such that it implements the operations $U_1^{u_1} \dots U_k^{u_k}$ iff the set of control qubits is in the basis state described by $|\psi_{c_1, \dots, c_i}\rangle = |1\rangle$ and $|\psi_{\bar{c}_1, \dots, \bar{c}_j}\rangle = |0\rangle$. For any other basis state, the CUG leaves the system of qubits unchanged. Figure 1 shows an example of the $C^1 \bar{C}^{3,6} U_1^2 U_2^4$ gate.

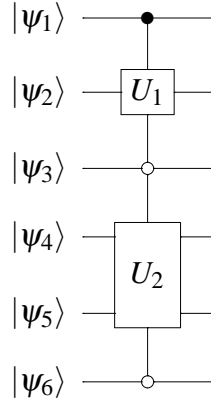


Figure 1: The $C^1 \bar{C}^{3,6} U_1^2 U_2^4$ gate, with C conditional on qubit 1 being $|1\rangle$, \bar{C} conditional on qubit 3 and 6 being $|0\rangle$, and the operations U_1 and U_2 are implemented on qubits 2 and 4 to 5 respectively.

3.1.2. Decomposition

An efficient way to evaluate arbitrary controlled unitary gates is to decompose the operation by defining the projection operators P_0 and P_1 as:

$$P_0 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, P_1 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

Note that P_0 and P_1 are non-unitary matrices and $P_0 + P_1 = I_2$ is the 2-by-2 identity matrix. Now consider the $C^1 U^2$ (abbreviated as CU) gate, shown in Figure 2.

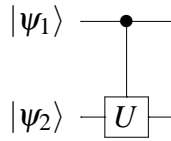


Figure 2: The CU gate.

It can be readily verified and proven that the matrix for the CU gate is given as $CU = P_0 \otimes I_2 + P_1 \otimes U$ (see appendix A for details). This result, called the decomposition of the CU gate as a sum, is graphically summarised in Figure 3.

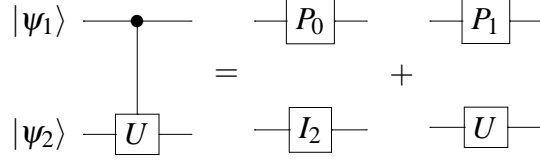


Figure 3: Decomposition of the CU gate.

The key idea is that we can use the projection operators P_0 and P_1 to project the set of control qubits to a basis state. For any basis state, the action of the CUG gate is either just the $U_1^{u_1} \dots U_k^{u_k}$ operators, or no action at all (i.e. the identity operator). By considering all possible basis states of the set of control qubits, we can construct the matrix of the CUG gate by summing together the action of the CUG gate corresponding to each possible basis state.

For any arbitrary $C^{c_1, \dots, c_i} U_1^{u_1} \dots U_k^{u_k}$ gate, consider replacing each conditional with a P_0 or P_1 operator. This can be done in 2^i distinct ways. For the basis state described by $|\psi_{c_1, \dots, c_i}\rangle = |1\rangle$, which corresponds to the permutation $C^m \rightarrow P_1 \forall m = c_1, \dots, c_i$, the action of the CUG is the operations $U_1^{u_1} \dots U_k^{u_k}$. Any other basis state (and hence permutation) corresponds to the action of the CUG being the identity operator. The sum of the 2^i permutations yields the matrix of the $C^{c_1, \dots, c_i} U_1^{u_1} \dots U_k^{u_k}$ gate. For example,

$$C^{1,3}U^2 = P_0 \otimes I_2 \otimes P_0 + P_0 \otimes I_2 \otimes P_1 + P_1 \otimes I_2 \otimes P_0 + P_1 \otimes U \otimes P_1,$$

as graphically shown in Figure 4 (see appendix B for a mathematical proof).

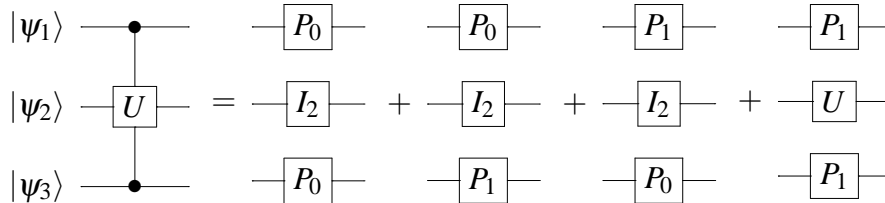


Figure 4: Decomposition of the $C^{1,3}U^2$ gate.

Similarly, for any arbitrary $\bar{C}^{\bar{c}_1, \dots, \bar{c}_j} U_1^{u_1} \dots U_k^{u_k}$ gate, consider the 2^j possible permutations that arise from replacing each \bar{C} conditional with a P_0 or P_1 operator. For the basis state described by $|\psi_{\bar{c}_1, \dots, \bar{c}_j}\rangle = |0\rangle$, which corresponds to the permutation $\bar{C}^n \rightarrow P_0 \forall n = \bar{c}_1, \dots, \bar{c}_j$, the action of the CUG is the operations $U_1^{u_1} \dots U_k^{u_k}$. Any other basis state corresponds to the action of the CUG being the identity operator. The sum of the 2^j permutations gives the matrix of the $\bar{C}^{\bar{c}_1, \dots, \bar{c}_j} U_1^{u_1} \dots U_k^{u_k}$ gate.

gate, for example,

$$\bar{C}^{1,3}U^2 = P_0 \otimes U \otimes P_0 + P_0 \otimes I_2 \otimes P_1 + P_1 \otimes I_2 \otimes P_0 + P_1 \otimes I_2 \otimes P_1,$$

as graphically shown in Figure 5.

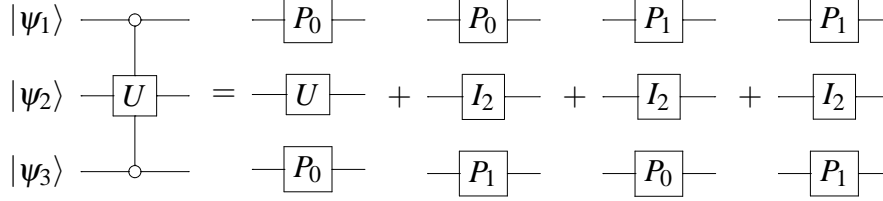


Figure 5: Decomposition of the $\bar{C}^{1,3}U^2$ gate.

Hence, for any arbitrary $C^{c_1, \dots, c_i} \bar{C}^{\bar{c}_1, \dots, \bar{c}_j} U_1^{u_1} \dots U_k^{u_k}$ gate, we consider each of the 2^{i+j} permutations that arise from replacing each C and \bar{C} conditional with a P_0 or P_1 operator. For the basis state described by $|\psi_{c_1, \dots, c_i}\rangle = |1\rangle$ and $|\psi_{\bar{c}_1, \dots, \bar{c}_j}\rangle = |0\rangle$, which corresponds to the permutation $C^m \rightarrow P_1 \forall m = c_1, \dots, c_i$ and $\bar{C}^n \rightarrow P_0 \forall n = \bar{c}_1, \dots, \bar{c}_j$, the action of the CUG is the operations $U_1^{u_1} \dots U_k^{u_k}$. Any other basis state corresponds to the action of the CUG being the identity operator. The sum of the 2^{i+j} permutations yields the matrix of the $C^{c_1, \dots, c_i} \bar{C}^{\bar{c}_1, \dots, \bar{c}_j} U_1^{u_1} \dots U_k^{u_k}$ gate. For example,

$$C^3 \bar{C}^1 U^2 = P_0 \otimes I_2 \otimes P_0 + P_0 \otimes U \otimes P_1 + P_1 \otimes I_2 \otimes P_0 + P_1 \otimes I_2 \otimes P_1,$$

as graphically shown in Figure 6.

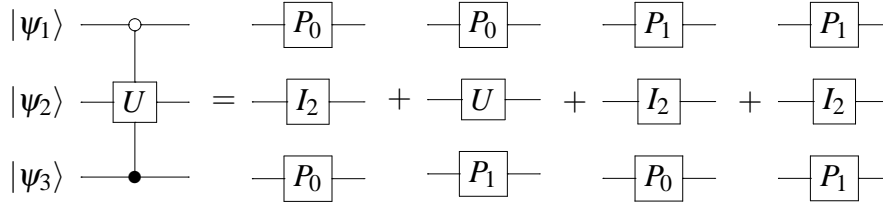


Figure 6: Decomposition of the $C^3 \bar{C}^1 U^2$ gate.

3.1.3. Reduction to its irreducible form

For an arbitrary $C^{c_1, \dots, c_i} \bar{C}^{\bar{c}_1, \dots, \bar{c}_j} U_1^{u_1} \dots U_k^{u_k}$ gate, a naive implementation of the previous section would require $2^{i+j} - 1$ matrix additions to compute the matrix of the gate. However, this overhead can be reduced significantly by noting that

only one permutation has the $U_1^{u_1} \dots U_k^{u_k}$ operators being implemented, while the other $2^{i+j} - 1$ possible permutations have identity operators substituted in for the $U_1^{u_1} \dots U_k^{u_k}$ operators. As an example, consider the gate (essentially the identity matrix I_8) shown in Figure 7, which has $2^2 - 1$ of the same permutations as in Figure 6. Consequently, we can write the decomposition of the $C^3 \bar{C}^1 U^2$ gate as the following

$$C^3 \bar{C}^1 U^2 = I_8 + P_0 \otimes U \otimes P_1 - P_0 \otimes I_2 \otimes P_1, \quad (1)$$

which is graphically represented by Figure 8.

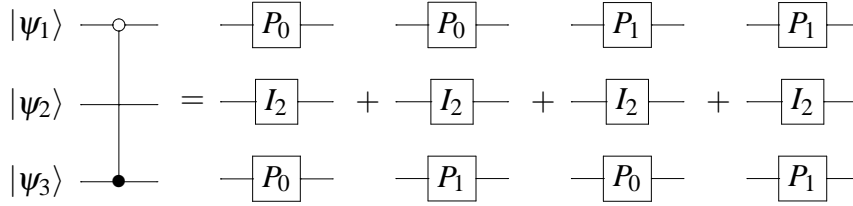


Figure 7: Decomposition of the I_8 gate.

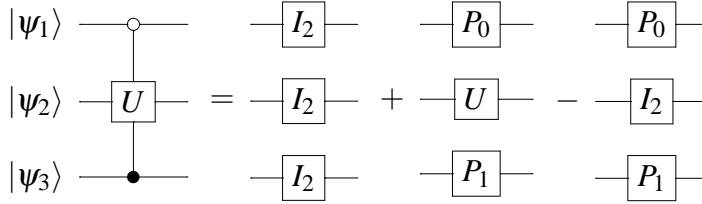


Figure 8: Optimized decomposition of the $C^3 \bar{C}^1 U^2$ gate.

For the general case, the matrix of any arbitrary $C^{c_1, \dots, c_i} \bar{C}^{\bar{c}_1, \dots, \bar{c}_j} U_1^{u_1} \dots U_k^{u_k}$ gate is simply the identity matrix (of appropriate dimensions), added together with the permutation that has the operations $U_1^{u_1} \dots U_k^{u_k}$ implemented, subtracted with the same permutation with the $U_1^{u_1} \dots U_k^{u_k}$ operators replaced with identity operators. In effect, the identity matrix is used to encapsulate $2^{i+j} - 1$ permutations. Hence, for any arbitrary $C^{c_1, \dots, c_i} \bar{C}^{\bar{c}_1, \dots, \bar{c}_j} U_1^{u_1} \dots U_k^{u_k}$ gate, computation of the gate matrix requires only two matrix additions, regardless of the number of controls or the gate dimensions. Note that the only instance in which this decomposition scheme is less efficient than the naive implementation is when only one C or \bar{C} conditional is involved. The optimized decomposition of a more complex example, the $C^2 \bar{C}^{1,4} U_1^3 U_2^5$ gate, is given in Figure 9.

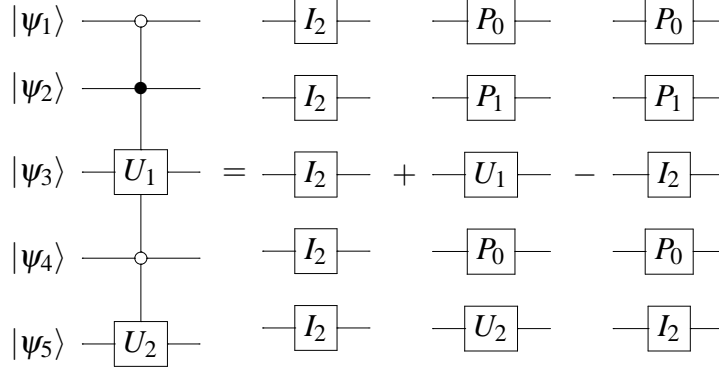


Figure 9: Optimized decomposition of the $C^2 \bar{C}^{1,4} U_1^3 U_2^5$ gate.

3.2. CUGs across qudits

3.2.1. Definitions and notation

Denote the wavefunction of the i -level qudit j as $|\psi_j^i\rangle$. Define a quantum circuit consisting of n qudits $\{|\psi_1^{\zeta_1}\rangle, |\psi_2^{\zeta_2}\rangle, \dots, |\psi_n^{\zeta_n}\rangle\}$ where ζ_i represents the number of levels in the i th qudit and $\zeta = \{\zeta_1, \zeta_2, \dots, \zeta_n\}$. We call ζ the quantum circuit profile, which is the list of qudit levels, arranged according to the order of the qudits. For example, any CUG applied across qubits has $\zeta = \{2, 2, \dots, 2\}$, since qubits are 2-level qudits. Also define $C_{s_i}^{c_i}$ as being conditional on the state $|s_i - 1\rangle$ of qudit c_i , where $1 \leq s_i \leq \zeta_{c_i}$.

A CUG with conditionals $C_{s_1}^{c_1} \dots C_{s_i}^{c_i}$ implementing unitary operations $U_1^{u_1} \dots U_k^{u_k}$, where u_1, \dots, u_k denotes the starting qudit of the corresponding U block, is represented by $C_{s_1}^{c_1} \dots C_{s_i}^{c_i} U_1^{u_1} \dots U_k^{u_k}$. Figure 10 shows an example of the $C_4^2 C_1^3 C_2^5 U_1^1 U_2^4$ gate.

3.2.2. Decomposition

We can readily extend the concept of projection operators to qudits, by defining $(P_{a,b})_{i,j} = \delta_{ai} \delta_{aj} \forall 1 \leq i, j \leq b$ (where δ_{ij} is the Kronecker delta) as the projection to the state $|a - 1\rangle$ acting on a b -leveled qudit, with the restriction $1 \leq a \leq b$. Hence every b -leveled qudit has a set of b projection operators defined, with the property $\sum_{a=1}^b P_{a,b} = I_b$.

For a general $C_{s_1}^{c_1} \dots C_{s_i}^{c_i} U_1^{u_1} \dots U_k^{u_k}$ gate, it is clear that by substituting each con-

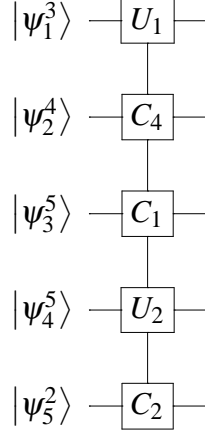


Figure 10: The $C_4^2 C_1^3 C_2^5 U_1^1 U_2^4$ gate, acting on 5 qudits of various levels. The quantum circuit profile is $\zeta = \{3, 4, 5, 5, 2\}$.

ditional with a (valid) projection operator, it would result in $\zeta_P = \prod_{j=1}^i \zeta_{c_j}$ distinct permutations. However, since the unitary operations $U_1^{u_1} \dots U_k^{u_k}$ are only carried out iff the control qudits c_1, \dots, c_i are in the states $|s_1 - 1\rangle, \dots, |s_i - 1\rangle$ respectively, only the permutation described by $C_{s_j}^{c_j} \rightarrow P_{s_j, \zeta_{c_j}} \forall j = 1, \dots, i$ exactly will have $U_1^{u_1} \dots U_k^{u_k}$ implemented; any other permutation will have identity operators substituted in place of $U_1^{u_1} \dots U_k^{u_k}$. The sum of all ζ_P permutations yields the matrix of the $C_{s_1}^{c_1} \dots C_{s_i}^{c_i} U_1^{u_1} \dots U_k^{u_k}$ gate. For example,

$$C_3^1 C_1^3 U^2 = P_{1,3} \otimes I_5 \otimes P_{1,2} + P_{1,3} \otimes I_5 \otimes P_{2,2} + P_{2,3} \otimes I_5 \otimes P_{1,2} + P_{2,3} \otimes I_5 \otimes P_{2,2} + P_{3,3} \otimes U \otimes P_{1,2} + P_{3,3} \otimes I_5 \otimes P_{2,2} \quad (2)$$

as graphically shown in Figure 11.

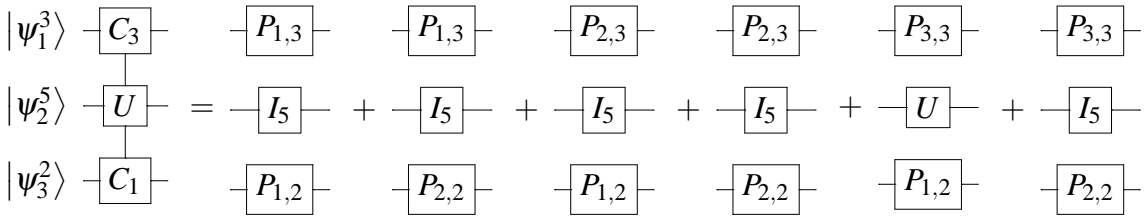


Figure 11: Decomposition of the $C_3^1 C_1^3 U^2$ gate.

3.2.3. Reduction to its irreducible form

As before, we can use the identity matrix (of appropriate dimensions) to encapsulate $\zeta_P - 1$ permutations of a $C_{s_1}^{c_1} \dots C_{s_i}^{c_i} U_1^{u_1} \dots U_k^{u_k}$ gate, since only one of the permutations have the $U_1^{u_1} \dots U_k^{u_k}$ operations implemented. The matrix of the CUG is thus the identity matrix (of appropriate dimensions) added together with the permutation described by $C_{s_j}^{c_j} \rightarrow P_{s_j, \zeta_{c_j}} \forall j = 1, \dots, i$, minus the same permutation with identity matrices substituted in place of $U_1^{u_1} \dots U_k^{u_k}$. The optimized decomposition of the $C_3^1 C_1^3 U^2$ gate is given in Figure 12. The optimized decomposition of a more complex example, the $C_4^2 C_1^3 C_2^5 U_1^1 U_2^4$ gate, is given in Figure 13.

$$\begin{array}{ccccccc}
 |\psi_1^3\rangle & -C_3- & & -I_3- & & -P_{3,3}- & & -P_{3,3}- \\
 & | & & & & & & \\
 |\psi_2^5\rangle & -U- & = & -I_5- & + & -U- & - & -I_5- \\
 & | & & & & & & \\
 |\psi_3^2\rangle & -C_1- & & -I_2- & & -P_{1,2}- & & -P_{1,2}-
 \end{array}$$

Figure 12: Optimized decomposition of the $C_3^1 C_1^3 U^2$ gate.

$$\begin{array}{ccccccc}
 |\psi_1^3\rangle & -U_1- & & -I_3- & & -U_1- & & -I_3- \\
 & | & & & & & & \\
 |\psi_2^4\rangle & -C_4- & & -I_4- & & -P_{4,4}- & & -P_{4,4}- \\
 & | & & & & & & \\
 |\psi_3^5\rangle & -C_1- & = & -I_5- & + & -P_{1,5}- & - & -P_{1,5}- \\
 & | & & & & & & \\
 |\psi_4^5\rangle & -U_2- & & -I_5- & & -U_2- & & -I_5- \\
 & | & & & & & & \\
 |\psi_5^2\rangle & -C_2- & & -I_2- & & -P_{2,2}- & & -P_{2,2}-
 \end{array}$$

Figure 13: Decomposition of the $C_4^2 C_1^3 C_2^5 U_1^1 U_2^4$ gate.

4. Comparison with the *QDENSITY* package

The *QDENSITY* package [1] provides many functions for the simulation of quantum circuits, two of which simulate the CNOT gate and the Toffoli gate. A more recent paper [7] introduces *QCWAVE* as an extension of the *QDENSITY* package. *QCWAVE* has the functions Op2 and Op3 that can be used to reproduce

the action of $CNOT_n$ and $Toffoli_n$ gates on state vectors, but does not give the matrix of the gates itself. We find it more straightforward and efficient to use the *QDENSITY* functions to construct the matrix and then act on the state vector, and hence we perform the following comparison using the *QDENSITY* package of version 4.0 (updated since [1]).

Here, we compare the CPU time taken to compute the matrices for the same gates, using the $CNOT$ and $Toffoli$ functions provided in the *QDENSITY* package and the more general $CUGate$ function provided in the *CUGates.m* package. The *QDENSITY* functions implements a decomposition using many more matrix additions and list manipulations in comparison with the scheme described in this paper.

We define the $CNOT_n$ gate as spanning n qubits with the C control located at the first qubit, and the NOT gate located at the n^{th} qubit. The $Toffoli_n$ gate is defined as spanning n qubits with the C controls at qubits 1 and 2, and the NOT gate located at the n^{th} qubit. Using these definitions, we are able to measure the CPU time taken to compute the matrix against n , which is plotted in Figure 14.

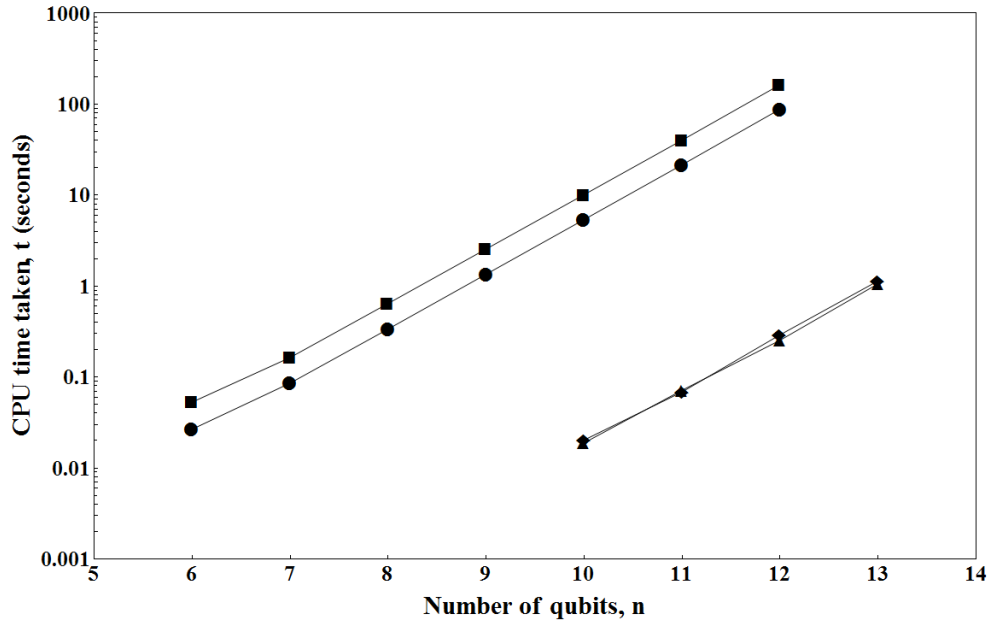


Figure 14: CPU time taken. Circle/Square: Time taken using the $CNOT$ / $Toffoli$ function in the *QDENSITY* 4.0 package. Diamond/Triangle: Time taken using the $CUGate$ function in the *CUGates.m* package with the sparse-matrix optimization to compute the $CNOT$ / $Toffoli$ gate.

As demonstrated in Figure 14, the CUGate function is significantly faster (by several orders of magnitude) than the CNOT and Toffoli functions provided in the QDENSITY package. In the actual *Mathematica* implementation of the CUGate function, we utilized sparse-matrix optimization to maximize calculation speed, which in this case, provides a speedup of about 1.8 compared to the CUGate function without the sparse-matrix optimization. It is also worth noting that while the Toffoli function takes almost twice as long as the CNOT function to compute its result, the CUGate function takes approximately the same length of time to compute the matrix of a CNOT and Toffoli gate for any particular n , which is expected from the decomposition scheme described in this paper. In general, computation of the matrix of any two controlled unitary gates spanning the same number of qubits using the CUGate function takes the same length of time.

To perform this analysis, we have timed the use of the functions in *Mathematica* using the Timing function, averaged over 10 trials. Computations were done on a laptop with an Intel Core i7-740QM processor with a speed of 1.73GHz. Results for $n < 10$ using the CUGate function is omitted since the minimum granularity of the Timing function is more than the CPU time needed for the CUGate function.

5. Worked examples

First load the CUGates.m package in *Mathematica* using the following syntax:

```
In[1] := Needs["CUGates"]
```

Brief descriptions of each function included in the CUGates.m package can be accessed using the '?' operator. For example,

```
In[2] := ?CUGate
```

```
Out[2] := CUGate[cpos,cbarpos,ubegin,umatrix]
```

Returns the matrix of a CUG across qubits with C conditionals at cpos, \bar{C} conditionals at cbarpos, and unitary operators umatrix with the corresponding starting positions ubegin.

```

In[3] := ?CUGateG
Out[3] := CUGateG[qcp,clist,ubegin,umatrix]

```

Returns the matrix of a CUG across qudits with conditionals described by clist, and unitary operators umatrix with the corresponding starting positions ubegin.

Note: clist is a list of {Index of qudit in qcp, Conditional state}

The qubit-specific subroutines are: BasisStateVector, CUGate, EqualSuperposition, HadamardGate, ListStates, MeasureQubits, MeasureSingleQubit, NOTGate, PHASEGate, SWAPGate and SWAPQubits.

The general qudit subroutines are: BasisStateVectorG, CUGateG, EqualSuperpositionG, ListStatesG, PHASEGateG, POP, QFTMinus, QFTPlus, RMinus, RPlus and SWAPQudits. The definitions for the functions QFTMinus and QFTPlus are similar to that of the QFT operator defined in [8].

5.1. Shor's algorithm

Figure 15 shows the implementation of Shor's algorithm to factorize $N = 15$ for co-prime, $C = 7$ [9].

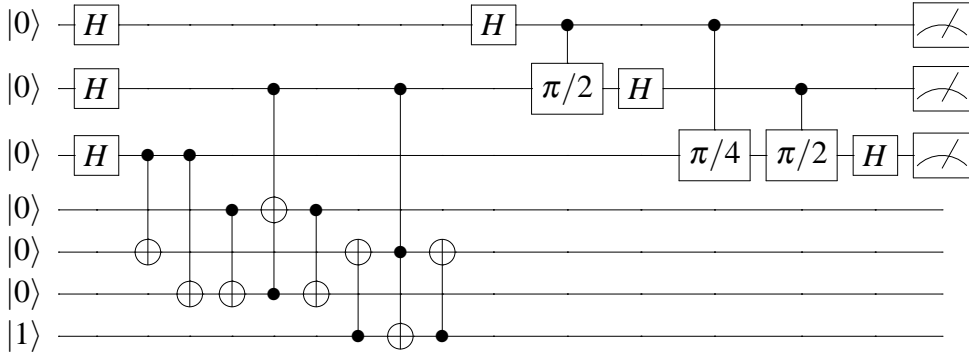


Figure 15: Quantum circuit for Shor's algorithm, $N = 15$ and $C = 7$.

Using *Mathematica*, we first initialize the qubit states as follows:

```

In[4] := InputVector = BasisStateVector[{0,0,0,0,0,0,1}];
HTransform = KroneckerProduct[HadamardGate[],HadamardGate[],
HadamardGate[],IdentityMatrix[2^4]];

```

Modular exponentiation is carried out on qubits 4 to 7 below:

```
In[5] := ModA = KroneckerProduct[IdentityMatrix[22],
      CUGate[{3},{},{5},{NOTGate[]}],IdentityMatrix[22]];
ModB = KroneckerProduct[IdentityMatrix[22],
      CUGate[{3},{},{6},{NOTGate[]}],IdentityMatrix[21]];
ModC = KroneckerProduct[IdentityMatrix[23],
      CUGate[{4},{},{6},{NOTGate[]}],IdentityMatrix[21]];
ModD = KroneckerProduct[IdentityMatrix[21],
      CUGate[{2,6},{},{4},{NOTGate[]}],IdentityMatrix[21]];
ModE = ModC;
ModF = KroneckerProduct[IdentityMatrix[24],
      CUGate[{7},{},{5},{NOTGate[]}]];
ModG = KroneckerProduct[IdentityMatrix[21],
      CUGate[{2,5},{},{7},{NOTGate[]}]];
ModH = ModF;
```

Next, the inverse QFT (Quantum Fourier Transform) is performed on the first three qubits:

```
In[6] := QftA = KroneckerProduct[HadamardGate[],IdentityMatrix[22]];
QftB = KroneckerProduct[CUGate[{1},{},{2},{PHASEGate[ $\pi/2$ ]}],
      IdentityMatrix[25]];
QftC = KroneckerProduct[IdentityMatrix[21],HadamardGate[],
      IdentityMatrix[25]];
QftD = KroneckerProduct[CUGate[{1},{},{3},{PHASEGate[ $\pi/4$ ]}],
      IdentityMatrix[24]];
QftE = KroneckerProduct[IdentityMatrix[21],
      CUGate[{2},{},{3},{PHASEGate[ $\pi/2$ ]}],IdentityMatrix[24]];
QftF = KroneckerProduct[IdentityMatrix[22],HadamardGate[],
      IdentityMatrix[24]];
```

We then multiply the matrices together from right to left, apply it to an initial qubit states, and obtain the final state of the quantum register.

```
In[7] := TMatrix = QftF.QftE.QftD.QftC.QftB.QftA.ModH.ModG.
          ModF.ModE.ModD.ModC.ModB.ModA.HTransform;
OutputVector = TMatrix.InputVector;
ListStates[OutputVector];
```

Out[7] := List of qubit states with a non-zero amplitude:

$$\begin{aligned} & \left(\frac{1}{4}\right) |0000001\rangle + \left(\frac{1}{4}\right) |0000100\rangle + \left(\frac{1}{4}\right) |0000111\rangle + \left(\frac{1}{4}\right) |0001101\rangle + \\ & \left(\frac{1}{4}\right) |0010001\rangle + \left(\frac{1}{4}\right) |0010100\rangle + \left(-\frac{1}{4}\right) |0010111\rangle + \left(-\frac{1}{4}\right) |0011101\rangle + \\ & \left(\frac{1}{4}\right) |0100001\rangle + \left(-\frac{1}{4}\right) |0100100\rangle + \left(\frac{i}{4}\right) |0100111\rangle + \left(-\frac{i}{4}\right) |0101101\rangle + \\ & \left(\frac{1}{4}\right) |0110001\rangle + \left(-\frac{1}{4}\right) |0110100\rangle + \left(-\frac{i}{4}\right) |0110111\rangle + \left(\frac{i}{4}\right) |0111101\rangle \end{aligned}$$

The most important part of the result is the state measurement of qubits 1, 2 and 3, which constitute the output register. Upon measurement, qubit 1 is solely in the computational basis $|0\rangle$, whereas qubits 2 and 3 are in a mixture of both computational bases, $|0\rangle$ and $|1\rangle$. Written in reverse order, we have a superposition of the combined states $|000\rangle$, $|010\rangle$, $|100\rangle$, and $|110\rangle$ for the three qubits in the output register, which has a periodicity of $p = 2$. According to Shor's algorithm, the factors are then given by the greatest common divisor (gcd) of $C^{\frac{2^{n-1}}{p}} \pm 1$ and N , where $n = 3$ is the number of qubits in the output register. Therefore $gcd(C^{\frac{2^{n-1}}{p}} \pm 1, N) = gcd(7^{\frac{2^{3-1}}{2}} \pm 1, 15) = gcd(7^2 \pm 1, 15) = 3, 5$, which are indeed the factors of $N = 15$.

5.2. Quantum random walks

Here, we are concerned with the quantum circuit implementation of quantum walks on highly symmetrical graphs. There exists different software packages that can implement quantum random walks across graphs, e.g. the QWalk package implements a quantum walk across 1-dimensional and 2-dimensional lattices [10] and the qwViz package visualize a quantum walks on arbitrarily complex graphs [11], as well as various quantum state based physical implementation schemes such as described in [12, 13], without reference to a circuit implementation of the graph. However, we consider circuit implementations here to illustrate the use of the CUGates package.

5.2.1. 16-length cycle

Consider the quantum circuit shown in Figure 16, which implements a quantum walk on a 16-length cycle using the Increment/Decrement gates [14] shown in Figure 17. First, we define the functions IncrementGate and DecrementGate in *Mathematica* as below to calculate the matrix of the Increment/Decrement gate, given the number of qubits involved.

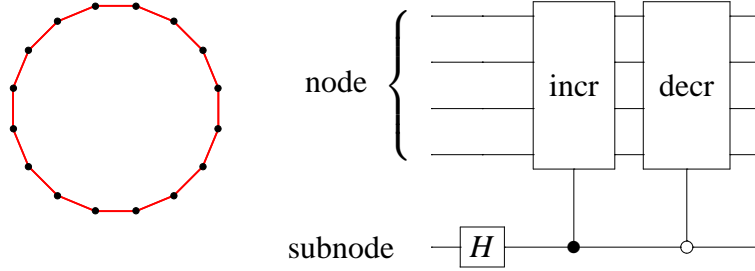


Figure 16: Quantum circuit implementing a quantum walk along a 16-length cycle.

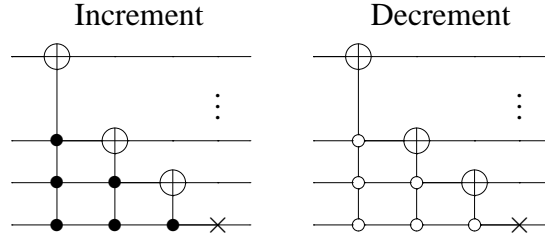


Figure 17: Increment and decrement gates on n qubits, producing cyclic permutations in the 2^n bit-string states.

```

In[8] := IncrementGate[NQubit_Integer] :=
(
Module
[ {ReturnMatrix,i,j},
ReturnMatrix = IdentityMatrix[ $2^{NQubit}$ ];
For[i = 1, i < NQubit, ++i,
ReturnMatrix = KroneckerProduct[IdentityMatrix[ $2^{i-1}$ ],
CUGate[Table[j,{j,i+1,NQubit}],{},{i},
{NOTGate[]}].ReturnMatrix;
];
Return[KroneckerProduct[IdentityMatrix[ $2^{NQubit-1}$ ],
NOTGate[]].ReturnMatrix];
]
)

```

```

In[9] := DecrementGate[NQubit_Integer] :=
(
Module
[ {ReturnMatrix,i,j},
ReturnMatrix = IdentityMatrix[ $2^{NQubit}$ ];
For[i = 1, i < NQubit, ++i,
ReturnMatrix = KroneckerProduct[IdentityMatrix[ $2^{i-1}$ ],
CUGate[{},{j,i+1,NQubit}],{i},
{NOTGate[]}].ReturnMatrix;
];
Return[KroneckerProduct[IdentityMatrix[ $2^{NQubit-1}$ ],
NOTGate[]].ReturnMatrix];
]
)

```


Using these definitions, we calculate the matrix of the circuit and apply it to the state vector signifying the initial vertex to be the 9th vertex (node representation of $|10000\rangle$) with the subnode initially set to $|0\rangle$.

```
In[10] := InputVector = BasisStateVector[{1,0,0,0,0}];
          Coin = KroneckerProduct[IdentityMatrix[2^4],HadamardGate[]];
          T1 = CUGate[{5},{},{1},{IncrementGate[4]}];
          T2 = CUGate[{},{5},{1},{DecrementGate[4]}];
          TMatrix = T2.T1.Coin;
          OutputVector = TMatrix.InputVector;
          ListStates[OutputVector];

Out[10] := List of qubit states with a non-zero amplitude:
           $\left(\frac{1}{\sqrt{2}}\right) |01100\rangle + \left(\frac{1}{\sqrt{2}}\right) |10011\rangle$ 
```

From the output, we can see that the initial state $|10000\rangle$ has been shifted to a superposition of states $|01100\rangle$ and $|10011\rangle$, which are the nodes adjacent to the initial state in a 16-length cycle. Further iterations will cause the quantum walk to propagate further along the cycle, with each state simultaneously moving to its adjacent states.

5.2.2. Complete 3^3 -graph with self-loops

As an example involving qudits in a quantum circuit, we analyze the quantum walk along the complete 3^n -graph with self loops as discussed in [14]. The complete 3^3 -graph with self-loops can be constructed as in Figure 18.

Here, the operator T_{\pm} is defined as $(T_{\pm})_{a,b} = \frac{1}{\sqrt{3}} e^{\pm \frac{2\pi i ab}{3}} \forall 1 \leq a, b \leq 3$, and the quantum circuit profile is now $\zeta = \{3, 3, 3, 3, 3, 3, 2\}$. This can be implemented in *Mathematica* as follows:

```
In[11] := TMinus = QFTMinus[3];
          TPlus = QFTPlus[3];
          QCProfile = {3,3,3,3,3,3,2};
```

The coin operator is calculated as follows:

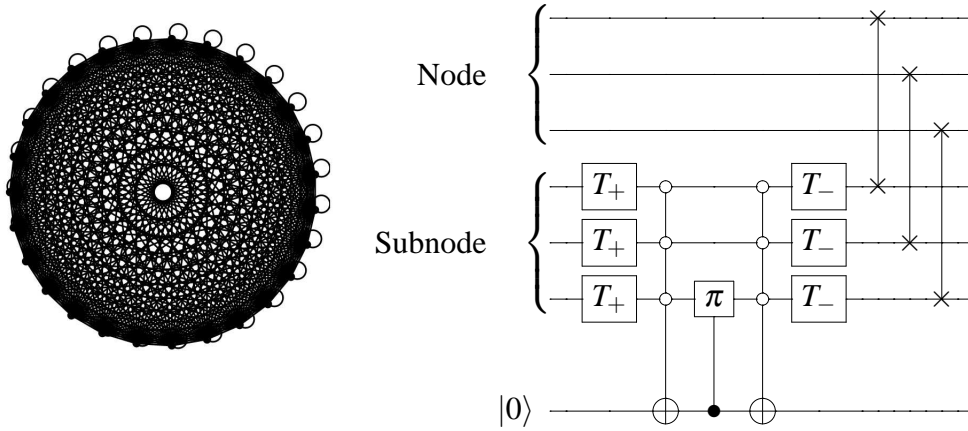


Figure 18: Quantum circuit implementing a quantum walk along a complete 3^3 -graph with self-loops. The node and subnode are composed of 3-level qudits (i.e. qutrits).

```

In[12] := C1 = SparseArray[KroneckerProduct[IdentityMatrix[33],
      TPlus,TPlus,TPlus,IdentityMatrix[2]]];
C2 = SparseArray[KroneckerProduct[IdentityMatrix[33],
      CUGateG[QCProfile,{ {4,1},{5,1},{6,1}},{7},{NOTGate[]},
      IdentityMatrix[2]]];
C3 = SparseArray[KroneckerProduct[IdentityMatrix[35],
      CUGateG[QCProfile,{ {7,1}},{6},{PHASEGateG[{π,0,0}1]}]]];
C4 = C2;
C5 = SparseArray[KroneckerProduct[IdentityMatrix[33],
      TMinus,TMinus,TMinus,IdentityMatrix[2]]];

```

The shifting operator can be implemented as such:

```

In[13] := T1 = SparseArray[KroneckerProduct[SWAPQudits[QCProfile,1,4],
      IdentityMatrix[32 * 2]]];
T2 = SparseArray[KroneckerProduct[IdentityMatrix[3],
      SWAPQudits[QCProfile,2,5], IdentityMatrix[3 * 2]]];
T3 = SparseArray[KroneckerProduct[IdentityMatrix[32],
      SWAPQudits[QCProfile,3,6], IdentityMatrix[2]]];

```

Finally, we can calculate the matrix of the circuit, and apply it to the state vector signifying the initial vertex to be the 1st vertex (node representation of $|000\rangle$), and obtain the result of a single iteration of the circuit.

```
In[14] := InputVector = BasisStateVectorG[QCProfile,{0,0,0,0,0,0}]
          TMatrix = Normal[T3.T2.T1.C5.C4.C3.C2.C1];
          OutputVector = TMatrix.InputVector;
          ListStatesG[QCProfile,OutputVector];

Out[14] := List of qudit states with a non-zero amplitude:
          ( $\frac{11}{27}$ )  $|0000000\rangle$  + ( $-\frac{16}{27}$ )  $|0010000\rangle$  + ( $-\frac{16}{27}$ )  $|0020000\rangle$  + ( $\frac{2}{27}$ )  $|0100000\rangle$  +
          ( $\frac{2}{27}$ )  $|0110000\rangle$  + ( $\frac{2}{27}$ )  $|0120000\rangle$  + ( $\frac{2}{27}$ )  $|0200000\rangle$  + ( $\frac{2}{27}$ )  $|0210000\rangle$  +
          ( $\frac{2}{27}$ )  $|0220000\rangle$  + ( $\frac{2}{27}$ )  $|1000000\rangle$  + ( $\frac{2}{27}$ )  $|1010000\rangle$  + ( $\frac{2}{27}$ )  $|1020000\rangle$  +
          ( $\frac{2}{27}$ )  $|1100000\rangle$  + ( $\frac{2}{27}$ )  $|1110000\rangle$  + ( $\frac{2}{27}$ )  $|1120000\rangle$  + ( $\frac{2}{27}$ )  $|1200000\rangle$  +
          ( $\frac{2}{27}$ )  $|1210000\rangle$  + ( $\frac{2}{27}$ )  $|1220000\rangle$  + ( $\frac{2}{27}$ )  $|2000000\rangle$  + ( $\frac{2}{27}$ )  $|2010000\rangle$  +
          ( $\frac{2}{27}$ )  $|2020000\rangle$  + ( $\frac{2}{27}$ )  $|2100000\rangle$  + ( $\frac{2}{27}$ )  $|2110000\rangle$  + ( $\frac{2}{27}$ )  $|2120000\rangle$  +
          ( $\frac{2}{27}$ )  $|2200000\rangle$  + ( $\frac{2}{27}$ )  $|2210000\rangle$  + ( $\frac{2}{27}$ )  $|2220000\rangle$ 
```

5.2.3. 3rd generation 3-Cayley tree

As a further example involving a mixture of qubits and qudits, we demonstrate how to implement a quantum walk on the 3rd generation 3-Cayley tree (shown in Figure 19) with the central node marked, by using its corresponding quantum circuit shown in Figure 20.

The G_n operator is defined as $(G_n)_{i,j} = \frac{2}{n} - \delta_{ij} \forall 1 \leq i, j \leq n$. Here, the G_3 operator acts on only 3 of the 4 subnode states, so it does not mix with the state $|11\rangle$. The \mathcal{R}_+ and \mathcal{R}_- gates are generalized increment and decrement gates respectively. For a b -leveled qudit, they are defined as b -by- b matrices given as $(\mathcal{R}_+)_{i,j} = \delta_{i(\bmod b)+1,j}$ and $(\mathcal{R}_-)_{i,j} = \delta_{i,j(\bmod b)+1}$ respectively. A natural extension to multiple qudits is given in Figure 21. In general, the R_R and R_L operators, shown in Figure 22, correspond to a clockwise and anticlockwise rotation of qudits. However, in the context of Figure 20, R_R and R_L are both single SWAP gates.

Given the length of the code needed to simulate the quantum circuit for a quantum walk along the 3-Cayley tree, we refer the reader to the *Mathematica* notebook *CUGates.nb*. The results of the quantum walk across 50 steps (starting

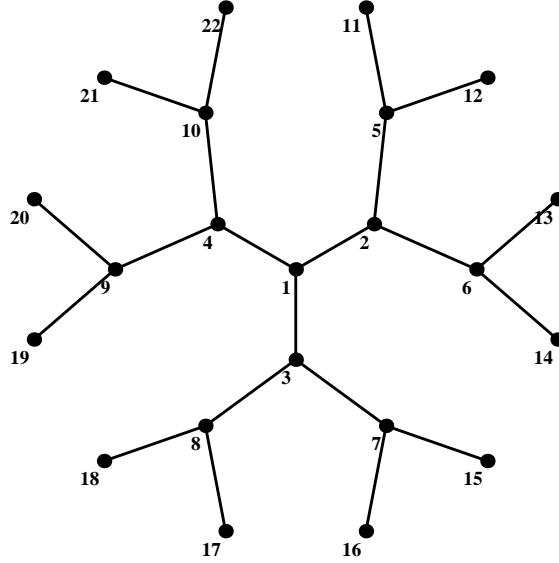


Figure 19: 3rd generation 3-Cayley tree.

in an equal superposition of vertex states, which is then subdivided according to the subnode states of the vertex) is shown in Figure 23, where the centre marked node is distinguished by its much larger probability peak.

6. Conclusions

The *Mathematica* notebook presented in this paper utilizes an irreducible form of matrix decomposition of a general controlled quantum gate with multiple conditionals and is highly efficient in simulating complex quantum circuits. It provides a powerful tool to assist researchers analyze the performance of proposed quantum circuits. It has helped to identify several errors in the quantum circuits described in [14], which was addressed and acknowledged in [15]. Another important application in which large and complex circuits need to be efficiently simulated is in the area of quantum error correction, in which generalized control unitary gates are used with both qubits and qudits [16, 17]. This package will prove to be immensely helpful in the design of codification circuits in this area. Implementation in *Mathematica* allows the code to be used in a cohesive and interactive environment which is nevertheless computationally powerful. The interactive nature of this environment also makes this notebook suitable for teaching, where

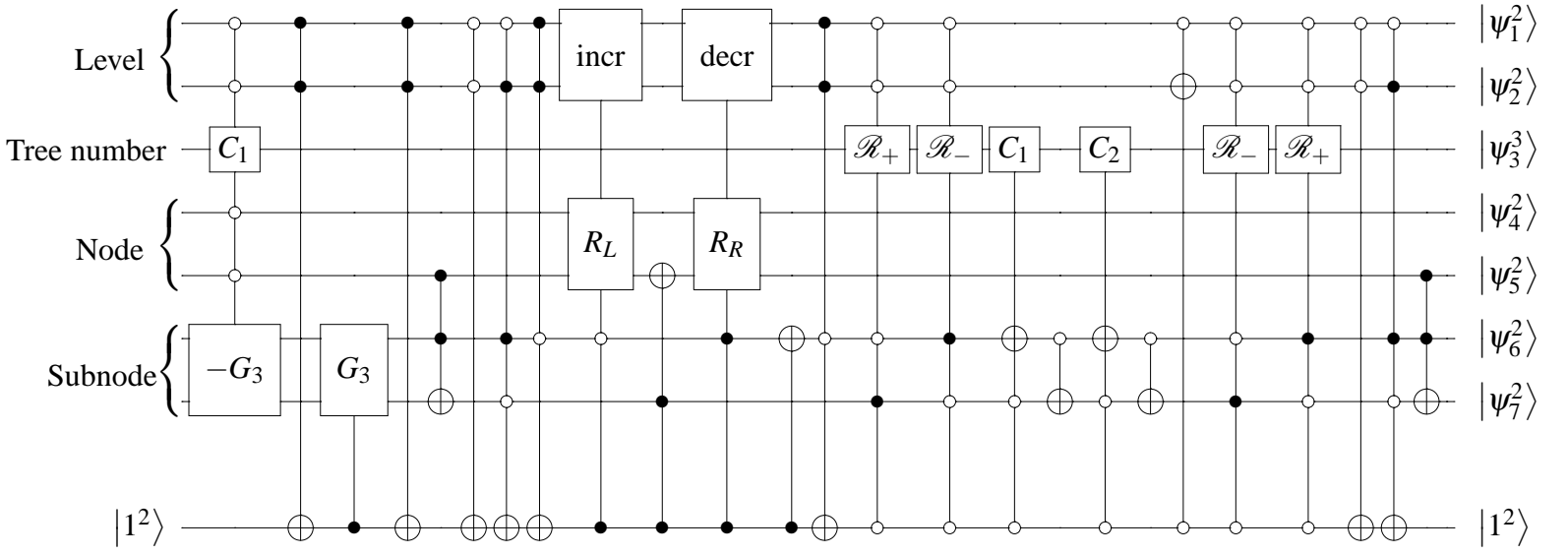


Figure 20: Quantum circuit implementing a quantum walk along a 3^{rd} generation 3-Cayley tree, with the central node marked. Any vertex is uniquely defined by a combination of the level, tree number, and node states.

quantum algorithms and quantum gate operations can be studied in detail.

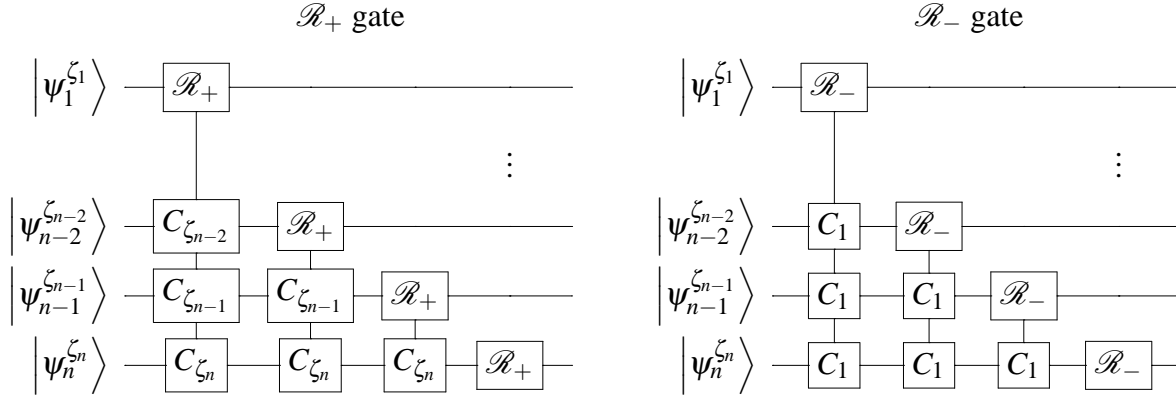


Figure 21: \mathcal{R}_+ and \mathcal{R}_- gates on n qudits, with a quantum circuit profile of $\zeta = \{\zeta_1, \zeta_2, \dots, \zeta_n\}$.

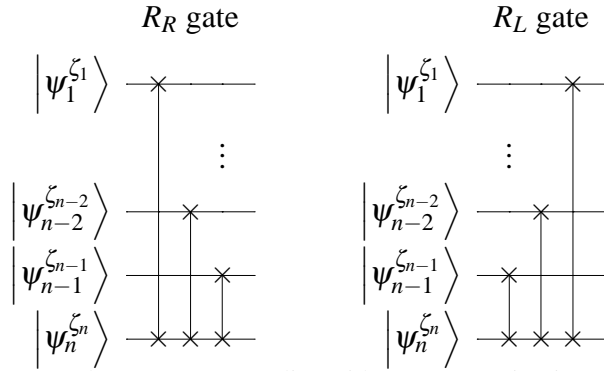


Figure 22: R_R and R_L gates on n qudits, with a quantum circuit profile of $\zeta = \{\zeta_1, \zeta_2, \dots, \zeta_n\}$.

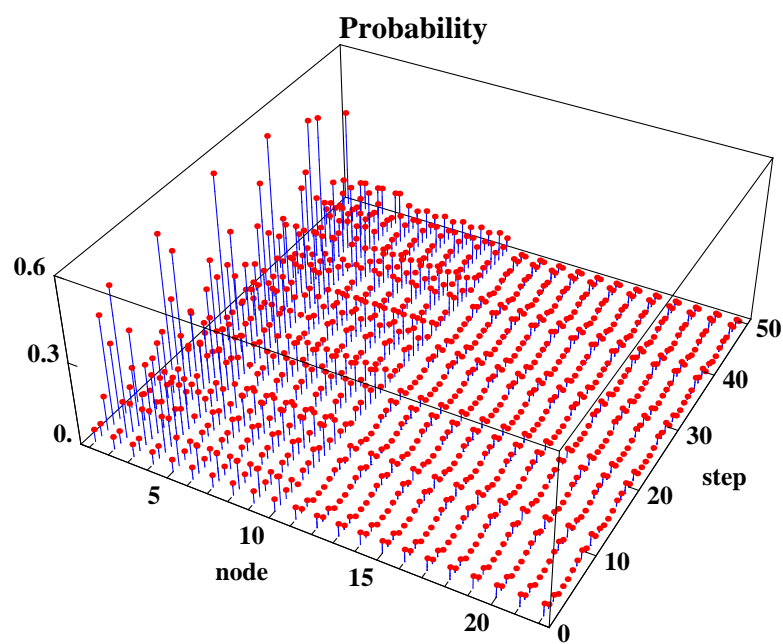


Figure 23: Probability distribution along the 3^{rd} generation 3-Cayley tree against the number of walking steps.

References

- [1] B. Julia-Diaz, J. M. Burdis, F. Tabakin, QDENSITY - a Mathematica quantum computer simulation, *Computer Physics Communications* 174 (2006) 914–934.
- [2] T. Radtke, S. Fritzsche, Simulation of n-qubit quantum systems: A computer-algebraic approach, *Computer Physics Communications* 173 (2005) 91–113.
- [3] K. M. Obenland, A. M. Despain, A parallel quantum computer simulator, at <http://arxiv.org/abs/quant-ph/9804039> (1998).
- [4] K. D. Raedt, K. Michielsen, H. D. Raedt, B. Trieu, G. Arnold, M. Richter, T. Lippert, H. Watanabe, N. Ito, Massively parallel quantum computer simulator, *Computer Physics Communications* 176 (2007) 121–136.
- [5] E. Gutierrez, S. Romero, M. A. Trenas, E. L. Zapata, Quantum computer simulation using the CUDA programming model, *Computer Physics Communications* 181 (2010) 283–300.
- [6] M. A. Nielsen, I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.
- [7] F. Tabakin, B. Julia-Diaz, QCWAVE - a Mathematica quantum computer simulation update, *Computer Physics Communications* (2011, in press).
- [8] A. S. Ermilov, V. E. Zobov, Implementation of the quantum order-finding algorithm by adiabatic evolution of two qudits, *Quantum Computers and Computing* 9 (2009) 39–48.
- [9] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, I. L. Chuang, Experimental realization of Shor’s quantum factoring algorithm using nuclear magnetic resonance, *Nature* 414 (2001) 883–887.
- [10] F. L. Marquezino, R. Portugal, The QWalk simulator of quantum walks, *Computer Physics Communications* 179 (2008) 359–369.
- [11] S. D. Berry, P. Bourke, J. B. Wang, qwviz: Visualisation of quantum walks on graphs, *Computer Physics Communications* 182 (2011) 2295.

- [12] K. Manouchehri, J. B. Wang, Quantum walks in an array of quantum dots, *Journal of Physics A* 41 (2008) 065304.
- [13] K. Manouchehri, J. B. Wang, Quantum random walks without walking, *Physical Review A* 80 (2009) 060304(R).
- [14] B. L. Douglas, J. B. Wang, Efficient quantum circuit implementation of quantum walks, *Physical Review A* 79 (2009) 052335.
- [15] B. L. Douglas, J. B. Wang, Erratum: Efficient quantum circuit implementation of quantum walks, *Physical Review A* 80 (2009) 059901(E).
- [16] R. Ionicioiu, T. P. Spiller, W. J. Munro, Generalized Toffoli gates using qudit catalysis, *Physical Review A* 80 (2009) 012312.
- [17] A. N. Al-Rabadi, Reversible viterbi algorithm and its closed-system q-domain circuit design and computation, *Journal of Circuits, Systems, and Computers* 18 (2009) 1627–1649.

Appendices

Appendix A. CU gate decomposition proof

For any arbitrary state, $P_0(a|0\rangle + b|1\rangle) \mapsto a|0\rangle$ and $P_1(a|0\rangle + b|1\rangle) \mapsto b|1\rangle$, i.e. the P_0 and P_1 operators projects arbitrary states onto the $|0\rangle$ and $|1\rangle$ computational basis state respectively. Consider the quantum circuit in Figure A.24, where $|\psi_1\rangle = a_1|0\rangle + b_1|1\rangle$ and $|\psi_2\rangle = a_2|0\rangle + b_2|1\rangle$.

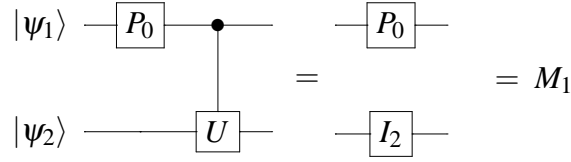


Figure A.24: Application of P_0 to the CU gate.

Since $P_0(|\psi_1\rangle) \mapsto a_1|0\rangle$, then

$$CU(P_0(|\psi_1\rangle) \otimes |\psi_2\rangle) \mapsto a_1|0\rangle \otimes |\psi_2\rangle \equiv P_0(|\psi_1\rangle) \otimes I_2(|\psi_2\rangle)$$

i.e. the U gate is not applied to the second qubit because the control qubit is in the state $a_1|0\rangle$ after the application of the P_0 operator, and thus the action of the CU gate is the identity operator. Hence, we can simplify the circuit, as shown in Figure A.24.

Similarly, if the P_1 operator is applied as in Figure A.25, then $P_1(|\psi_1\rangle) \mapsto b_1|1\rangle$ and thus

$$CU(P_1(|\psi_1\rangle) \otimes |\psi_2\rangle) \mapsto b_1|1\rangle \otimes U(|\psi_2\rangle) \equiv P_1(|\psi_1\rangle) \otimes U(|\psi_2\rangle),$$

because the control qubit is in the state $b_1|1\rangle$ after the application of the P_1 operator, so the action of the CU gate is the U^2 operator. The equivalent circuit is also shown in Figure A.25.

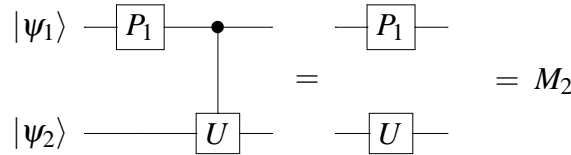


Figure A.25: Application of P_1 to the CU gate.

Note that M_1 and M_2 , as defined in Figures A.24 and A.25 respectively, are non-unitary. However the sum $M_1 + M_2 = P_0 \otimes I_2 + P_1 \otimes U$ is unitary and also

$$M_1 + M_2 = CU (P_0 \otimes I_2) + CU (P_1 \otimes I_2) \quad (\text{A.1})$$

$$= CU ((P_0 + P_1) \otimes I_2) \quad (\text{A.2})$$

$$= CU (I_2 \otimes I_2) \quad (\text{A.3})$$

$$= CU$$

Consequently, $CU = P_0 \otimes I_2 + P_1 \otimes U$.

Appendix B. $C^{1,3}U^2$ gate decomposition proof

The decomposition can be derived by considering each of the possible permutations, which are defined as follows:

$$M_1 = C^{1,3}U^2 (P_0 \otimes I_2 \otimes P_0) = P_0 \otimes I_2 \otimes P_0 \quad (\text{B.1})$$

$$M_2 = C^{1,3}U^2 (P_0 \otimes I_2 \otimes P_1) = P_0 \otimes I_2 \otimes P_1 \quad (\text{B.2})$$

$$M_3 = C^{1,3}U^2 (P_1 \otimes I_2 \otimes P_0) = P_1 \otimes I_2 \otimes P_0 \quad (\text{B.3})$$

$$M_4 = C^{1,3}U^2 (P_1 \otimes I_2 \otimes P_1) = P_1 \otimes U \otimes P_1$$

As before, we consider the permutation sum :

$$M_1 + M_2 + M_3 + M_4 = C^{1,3}U^2 (P_0 \otimes I_2 \otimes P_0) + C^{1,3}U^2 (P_0 \otimes I_2 \otimes P_1) + \quad (\text{B.4})$$

$$C^{1,3}U^2 (P_1 \otimes I_2 \otimes P_0) + C^{1,3}U^2 (P_1 \otimes I_2 \otimes P_1) \quad (\text{B.5})$$

$$= C^{1,3}U^2 (P_0 \otimes I_2 \otimes (P_0 + P_1)) + \quad (\text{B.6})$$

$$P_1 \otimes I_2 \otimes (P_0 + P_1)) \quad (\text{B.7})$$

$$= C^{1,3}U^2 ((P_0 + P_1) \otimes I_2 \otimes I_2) \quad (\text{B.8})$$

$$= C^{1,3}U^2 (I_2 \otimes I_2 \otimes I_2) \quad (\text{B.9})$$

$$= C^{1,3}U^2$$

Consequently, $C^{1,3}U^2 = P_0 \otimes I_2 \otimes P_0 + P_0 \otimes I_2 \otimes P_1 + P_1 \otimes I_2 \otimes P_0 + P_1 \otimes U \otimes P_1$.

Appendix C. Arbitrary CUG decomposition

For an arbitrary CUG across qubits with k conditionals, we have 2^k possible permutations when placing a P_0 or P_1 projection operator in front of each conditional. Each permutation then has a column that is described by the tensor product of the projection operators with identity matrices in the appropriate positions

placed in front of the CUG. Proving that the sum of these permutations is equal to the gate itself is fairly trivial; it simply involves factoring together permutations that differ by a single conditional, using the identity $P_0 + P_1 = I_2$, and then doing so repeatedly until we end up with the original CUG. The simplification comes by considering the action of the projection operators on the state going into the CUG, and since the CUG implements the action iff the input state is in the basis state corresponding to the conditionals, we can easily work out which of the permutations has the action of the CUG implemented, while the rest do not.

Similarly, for an arbitrary CUG across qudits, we have a number of permutations corresponding to the qudit levels on which the conditionals are placed, and by using the identity $\sum_{a=1}^b P_{a,b} = I_b$, we can readily prove that the sum of all permutations corresponds to the CUG itself, and can thus simplify the permutations as before. In both cases, we can simplify the decomposition considerably by using the identity matrix to represent the sum of all permutations with no action applied, then adding on the appropriate permutation with the action of the CUG and subtracting the same permutation without the action.

